**Problem: Minimum Spanning Tree with Constraints**

**Problem Statement**

You are given a city with n junctions connected by mmm bidirectional roads, each with a certain cost to build. Your task is to construct a minimum spanning tree (MST) that connects all the junctions with the minimum cost. However, there is a constraint: some junctions must be connected directly. These are critical connections that must be included in the MST.

**Scenario**

The city government wants to ensure certain key areas are directly connected while still minimizing the overall construction cost. As an engineer, you need to incorporate these mandatory connections into your MST while ensuring the total cost remains minimal.

**Input Format**

* The first line contains two integers n and m, representing the number of junctions and the number of roads, respectively.
* The next m lines contain three integers u, v, and w each, indicating a bidirectional road between junction u and junction v with a cost w.
* The next line contains an integer k, representing the number of mandatory connections.
* The next k lines contain two integers a and b each, indicating that there must be a direct road between junction a and junction b in the MST.

**Constraints**

* 2 <= n <= 100
* 0 <= m <= n×(n−1)/2
* 1 <= w <= 1000
* 0 <= u, v, a, b < n
* u != v
* a !=b
* All roads are unique.

**Output Format**

* Print an integer denoting the minimum cost to construct the MST with the given constraints. If it is not possible to construct such an MST, print -1.

**Sample Input**

6 9

0 1 1

0 2 4

0 3 3

1 2 2

1 4 7

2 4 5

2 5 9

3 4 6

4 5 8

2

1 2

3 4

**Sample Output**

20

**Explanation**

The minimum spanning tree must include the edges (1, 2) and (3, 4). The MST with the minimum cost satisfying these constraints has a total cost of 26.

**Additional Test Cases**

**Test Case 1**

4 5

0 1 1

0 2 3

1 2 2

1 3 4

2 3 5

1

0 1

**Output:**

7

**Test Case 2**

5 7

0 1 2

0 2 3

1 2 1

1 3 5

2 3 4

2 4 7

3 4 6

2

0 1

2 3

**Output:**

13

**Test Case 3**

3 3

0 1 5

1 2 10

0 2 7

1

0 2

**Output:**

12

**Test Case 4**

4 4

0 1 3

1 2 5

2 3 6

3 0 4

2

0 1

2 3

**Output:**

13

**Test Case 5**

6 8

0 1 10

0 2 15

1 2 5

1 3 7

2 3 8

2 4 10

3 4 12

4 5 6

3

0 1

1 3

4 5

**Output:**

38

**Solution**

To solve this problem, we can use Kruskal's algorithm with Union-Find (Disjoint Set Union) to construct the MST. We'll first ensure the mandatory connections are included and then build the MST with the remaining edges.

Here's the solution in Python:

python

class UnionFind:

def \_\_init\_\_(self, n):

self.parent = list(range(n))

self.rank = [1] \* n

def find(self, u):

if self.parent[u] != u:

self.parent[u] = self.find(self.parent[u])

return self.parent[u]

def union(self, u, v):

root\_u = self.find(u)

root\_v = self.find(v)

if root\_u != root\_v:

if self.rank[root\_u] > self.rank[root\_v]:

self.parent[root\_v] = root\_u

elif self.rank[root\_u] < self.rank[root\_v]:

self.parent[root\_u] = root\_v

else:

self.parent[root\_v] = root\_u

self.rank[root\_u] += 1

return True

return False

def kruskal\_with\_constraints(n, edges, mandatory):

uf = UnionFind(n)

mst\_cost = 0

# Include all mandatory edges

for u, v in mandatory:

if uf.union(u, v):

mst\_cost += next(w for (x, y, w) in edges if (x == u and y == v) or (x == v and y == u))

# Sort edges by weight

edges.sort(key=lambda x: x[2])

# Kruskal's algorithm

for u, v, w in edges:

if uf.union(u, v):

mst\_cost += w

# Check if all nodes are connected

root = uf.find(0)

if all(uf.find(i) == root for i in range(n)):

return mst\_cost

else:

return -1

# Input

n, m = map(int, input().strip().split())

edges = [tuple(map(int, input().strip().split())) for \_ in range(m)]

k = int(input().strip())

mandatory = [tuple(map(int, input().strip().split())) for \_ in range(k)]

# Output

print(kruskal\_with\_constraints(n, edges, mandatory))